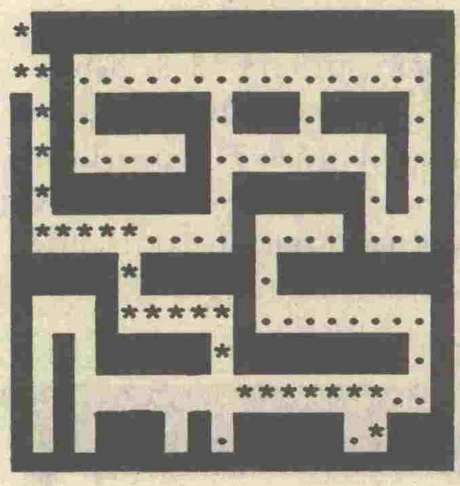
จเร เลิศสุดวิชัย

สร้างแพคเกจสแต็ก
จากแพคเกจลิงค์ลิสต์

นำเอาแพคเกจลิงค์ลิสต์มาสร้างเป็นแพคเกจสแต็ก พร้อมทั้งตัวอย่างปัญหาที่ต้องใช้สแต็กช่วยในการแก้ไข



ในตอนที่แล้วเราได้สร้างแพคเกจคิวจากแพคเกจลิงค์ลิสต์ไปแล้ว โดยที่เราไม่ต้องทำการปรับแต่งแพคเกจลิงค์ลิสต์เลย ในฉบับนี้ก็เช่นกัน จะเป็นการนำเอาแพคเกจลิงค์ลิสต์มาสร้างเป็นแพคเกจสแต็กพร้อมทั้งตัวอย่างปัญหาที่ต้องใช้สแต็กช่วยในการแก้ไขปัญหา เพื่อแสดงวิธีการนำแพคเกจสแต็กไปใช้งาน

แพคเกจสแต็ก

คุณสมบัติของสแต็กจะตรงกันข้ามกับคิว การดึงข้อมูลออกจากคิวนั้นลำดับข้อมูลจะได้ตรงกันกับลำดับของข้อมูลที่เข้า แต่สำหรับสแต็กข้อมูลที่ถูกนำออกนั้นลำดับของข้อมูลจะกลับกับลำดับของข้อมูลที่ถูกนำเข้าสแต็ก ซึ่งมักเรียกคุณสมบัติของสแต็กว่า "LIFO" หรือ "Last In First Out" นั้นเอง

จากคุณสมบัติที่แตกต่างกันระหว่างสแต็กกับคิว แต่การที่เราสร้างคิวนั้นเราสามารถนำเอาข้อมูลเก็บในลิงค์ลิสต์โดยทำตามคุณสมบัติของคิว ในกรณีของสแต็กเช่นเดียวกันเราจะใช้ลิงค์ลิสต์ในการเก็บข้อมูล การเพิ่มข้อมูลจะทำโดยเพิ่มข้อมูลเข้าไปที่ส่วนหัวของลิสต์ แต่การนำข้อมูลออกก็จะนำข้อมูลออกที่หัวเช่นกัน (ต่างกับคิวที่นำข้อมูลเข้าทางหัวของลิสต์ แต่การนำข้อมูลออกจะนำออกทางท้ายลิสต์แทน) และเช่นเดียวกับคิวในสแต็กก็จะมีฟังก์ชันพื้นฐานสำหรับการจัดการข้อมูล คือฟังก์ชัน Stpush( ) และ Stpop( ) ส่วนฟังก์ชันอื่นที่ใช้ในการติดตั้งค่าเริ่มต้นของสแต็ก ก็จะใช้ฟังก์ชันบางส่วนของลิงค์ลิสต์เองได้ แสดงดังในรูปที่ 1 ซึ่งเป็นไฟล์ส่วนหัวสำหรับแพคเกจสแต็ก ชื่อ Stackdef.h

จากรูปที่ 1 จะพบว่ามีการกำหนดชื่อฟังก์ชันและแบบของข้อมูลใหม่ให้สอดคล้องกับแพคเกจสแต็ก ซึ่งความจริงก็อาศัยฟังก์ชันของแพคเกจลิงค์ลิสต์นั่นเอง ดังเช่นฟังก์ชัน Stitemsize () ซึ่งเป็นฟังก์ชันสำหรับกำหนดขนาดของข้อมูลที่จัดเก็บในแต่ละช่องเก็บข้อมูลของสแต็กซึ่งใช้ฟังก์ชัน LLsetsize( ) กำหนดขนาดของข้อมูลในแต่ละของลิสต์นั่นเอง ส่วนฟังก์ชันของสแต็กที่เหลือคือฟังก์ชัน Stset( ) ใช้กำหนดตัวแปรสแต็กที่จะใช้งานก็เป็นฟังก์ชัน LLsetlist () และสุดท้ายฟังก์ชัน Stlength () ซึ่ง

```
/* stackdef.h */
#include "lldef.h"

#define Stset          LLsetlist     /* set current stack */
#define Stitemsize     LLsetsize     /* define size of item in stack */
#define Stlength       LLlength      /* find number of item in stack */

#define STACK          LINKLIST

Stpush (char *newitem);
Stpop (char *olditem);
```

รูปที่ 1 ไฟล์ stackdef.h

ใช้หาจำนวนข้อมูลในสแต็กว่ามีจำนวนเท่าไร ก็เป็นฟังก์ชัน LLlength () ของแพคเกจลิงค์ลิสต์

สำหรับตัวโมดูลของแพคเกจสแต็ก ซึ่งแสดงไว้ในรูปที่ 2 ภายในโมดูลประกอบด้วยฟังก์ชันพื้นฐานของสแต็ก 2 ฟังก์ชันคือ ฟังก์ชัน Stpush () และฟังก์ชัน Stpop () โดยถ้าสังเกตจะพบว่าฟังก์ชันทั้งสองจะเรียกใช้ฟังก์ชันของแพคเกจลิงค์ลิสต์อีกชั้นหนึ่ง ตัวอย่างเช่น ฟังก์ชัน Stpush () เป็นฟังก์ชันที่นำเอาข้อมูลเข้าไปเก็บสแต็ก ซึ่งคือการเก็บข้อมูลเข้าที่ส่วนหัวของลิสต์นั่นเอง เช่นเดียวกันกับฟังก์ชัน Stpop () เป็นการนำข้อมูลออกสแต็กโดยการนำข้อมูลออกจากทางส่วนหัวลิสต์นั่นเอง

การประยุกต์ใช้งานแพคเกจสแต็ก

การนำเอาแพคเกจสแต็กไปใช้งานก็เช่นเดียวกับแพคเกจลิงค์ลิสต์และแพคเกจคิว คือจะต้องประกาศใช้ตัวแปรที่จะทำหน้าที่เป็นสแต็กขึ้นมาก่อน จากนั้นจึงจะเรียกฟังก์ชัน Stset () เพื่อบ่งบอกให้แพคเกจสแต็กทราบว่าในโมดูลสแต็กจะต้องกระทำตัวแปรสแต็กตัวใดโดยส่งผ่านตัวแปรสแต็กที่ต้องการจะใช้งานให้กับฟังก์ชัน Stset () ด้วย หลังจากนั้นจึงจะกำหนดขนาดของข้อมูลที่จะบรรจุลงในสแต็กแต่ละตัวด้วยฟังก์ชัน Stitemsize () และเมื่อกำหนดค่าต่าง ๆ ให้กับแพคเกจสแต็กเรียบร้อยแล้วก็จะสามารถนำข้อมูลเข้าและออกจากสแต็กได้ด้วยฟังก์ชัน Stpush () และฟังก์ชัน Stpop () และอย่าลืมเมื่อต้องการใช้สแต็กตัวอื่น (ในกรณีที่มีการใช้งานสแต็กหลาย ๆ ตัวในโปรแกรมเดียวกัน) จะต้องใช้ฟังก์ชัน Stset () เพื่อเลือกตัวแปรสแต็กที่จะใช้งานใหม่ด้วยทุกครั้ง

```
/* Stack Module: stack.c */

#include <stdio.h>
#include "stackdef.h"

/* Pointer to currently active stack */
static struct STACK *st;

Stpush (newitem)
char *newitem;
{
    if (LLlength() == 0)
       LLinit (newitem);
    else
       LLaddhead (newitem);
}

Stpop (olditem)
char *olditem;
{
    LLretrieve (olditem);
    LLdelete ();
}
```

รูปที่ 2 โมดูลของแพคเกจ stack

เพื่อเสริมความเข้าใจจึงยกตัวอย่างปัญหาที่จะนำเอาแพคเกจสแต็กมาใช้ช่วยแก้ไขปัญหา ซึ่งตัวอย่างโปรแกรมการหาทางออกจากเขาวงกต (MAZE)

โปรแกรมหาทางออกจากเขาวงกต

ปัญหาเรื่องเขาวงกตเกิดขึ้นมาจากในห้องทดลอง ซึ่งต้องการทดสอบเกี่ยวกับการเรียนรู้ของหนู การทดลองทำโดยนำเอาหนูมาปล่อยที่ทางเข้าของกล่องที่มีฝาปิด และภายในกล่องถูกสร้างเส้นทางที่วกวนโดยตั้งเป็นกำแพงกั้นตามเส้นทาง และจะมีทางออกจากกล่องดังกล่าวเพียงทางเดียว นักวิทยาศาสตร์ได้ทำการศึกษาการเรียนรู้ของหนูด้วยวิธีนี้มานานจนในยุคของคอมพิวเตอร์ก็ได้มีคน (แน่นอนต้องเป็นมนุษย์ไม่ใช่หนู) คิดที่จะหาทางเดินออกจากเขาวงกตด้วยเครื่องมือคือคอมพิวเตอร์

เราจะตัดส่วนที่เกี่ยวกับการสร้างทางเขาวงกตออกไป และพิจารณากัน

เฉพาะการหาเส้นเดินที่จะออกจากเขาวงกตแทน มาดูแนวความคิดคร่าว ๆ กันก่อนว่าเราจะมีวิธีการที่จะหาทางออกจากเขาวงกตได้อย่างไร คิดแบบง่าย ๆ (ไม่ซับซ้อนถึงขนาดที่ต้องหาเส้นทางออกที่สั้นที่สุด) โดยลองสมมติว่าตัวเราเป็นหนูแทน และเริ่มเดินเข้าไปเขาวงกต ในเส้นทางของเขาวงกตจะมีทางที่ซับซ้อน มีทั้งทางแยก ทางตันอยู่จำนวนมาก ถ้าเราคิดแบบง่ายในกรณีที่เดินแล้วพบทางตัน เราจะต้องเดินย้อนถอยหลังกลับตามเส้นเดิม จนกระทั่งถึงจุดที่ทางแยก ซึ่งเราจะต้องจำได้ว่าทางแยก ณ จุดนั้นทางใดบ้างที่เราเคยเดินไปแล้ว โดยถ้าในมือของเราถือชอล์กเขียนกระดาน ณ จุดทางแยกนั้นก่อนจะเดินไปในทิศทางที่เลือก เราจะเขียนทิศทางที่เราได้เลือกเดินไปแล้ว ซึ่งทิศทางที่เราเขียนบันทึกเอาไว้นั้นจะช่วยบ่งบอกเราอีก ถ้าหากว่าทิศทางเลือกเดินไปแล้วนั้นเป็นทางตัน เมื่อย้อนถอยกลับมาอีกครั้งจะทราบได้ว่าทิศใดบ้างที่ยังไม่ได้เลือกเดิน ซึ่งถ้าเราใช้วิธีนี้ทำเช่นเดียวกับทุก ๆ จุดที่เราผ่าน สุดท้ายก็จะสามารถค้นหาทางออกได้

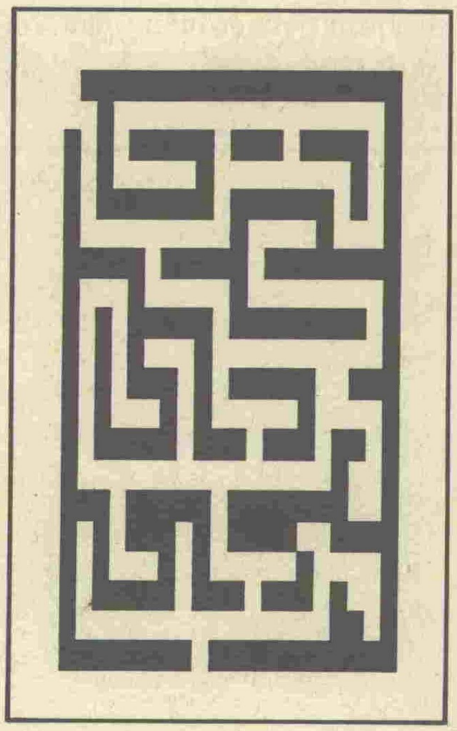
วิธีการที่กล่าวข้างต้นเป็นวิธีการหาทางออกจากเขาวงกตแบบง่าย ๆ วิธีหนึ่ง ซึ่งความจริงแล้วมีอัลกอลิทึมอยู่หลายวิธี ซึ่งจะไม่ขอกล่าว ณ ที่นี้

จากวิธีการหาทางออกจากเขาวงกตวิธีนี้ จะสังเกตเห็นว่าก่อนจะเลือกเดินไปในเส้นทางหรือจุดใดนั้น จะต้องทำการบันทึกและจดจำข้อมูลสภาพของจุดนั้น ๆ เอาไว้ เพื่อเอาไว้เป็นข้อมูลในการตัดสินใจเมื่อมีการย้อนถอยกลับมาจุดจุดนั้นอีกครั้งหนึ่ง และจากลักษณะการเก็บข้อมูลสถานะของจุดต่าง ๆ รวมทั้งการดึงข้อมูลสถานะของจุด เมื่อมีการย้อนถอยหลังจะตรงกับวิธีการเก็บและดึงข้อมูลจากสแต็ก ดังนั้นปัญหานี้จึงสามารถใช้สแต็กมาช่วยแก้ไขได้

หลังจากการเดินวิธีดังกล่าวแล้วในที่สุดก็จะพบทางออกจากเขาวงกต และผลพลอยได้ที่สำคัญคือ จุดต่าง ๆ ที่เคยผ่านโดยไม่มีการย้อนกลับ จะยังคงค้างอยู่ในสแต็กที่ใช้เก็บเอาไว้ นั่นคือเส้นทางที่เป็นเส้นที่ถูกต้องในการเดินหาทางออกจากเขาวงกต

สร้างเขาวงกตในโปรแกรม

ในโปรแกรมจะแทนเขาวงกตด้วยอะเรย์ขนาด 2 ทิศทางของตัวอักษร ชื่อ maze ซึ่งมีขนาดเท่ากัน MAXROW × MAXCOL (มีขนาดไว้ 20 × 20 แต่กำหนดเกินเอาไว้ 20 × 21 เพื่อใส่อักษร NULL เอาไว้ในแต่ละแถว) และการสร้างเขาวงกตจะทำโดยการ



รูปที่ 3 ตัวอย่างทางเดินในเขาวงกต

อ่านข้อมูลมาจากไฟล์ maze.dat โดย ข้อมูลจะประกอบไปด้วย ช่องว่าง ซึ่งหมายถึงทางเดินและกำแพงซึ่งแทนด้วยตัวอักษรรหัสแอสกี้ 219 ทั้ง 20 บรรทัด บรรทัดละ 20 ตัวอักษร การเขียนข้อมูลหาการสร้างเขาวงกตในไฟล์ maze.dat จะมีข้อกำหนดดังนี้

1. ทางเข้าเขาวงกตจะอยู่ที่ตำแหน่ง (1, 1) เท่านั้น (ถ้าจะเปลี่ยนจะต้องแก้ไขในโปรแกรมด้วย)

2. ทางออกจากเขาวงกตจะอยู่บริเวณส่วนล่างเท่านั้น คือจะอยู่ในบรรทัดที่ 20

3. ขอบนอกของเขาวงกต ยกเว้นทางเข้าและทางออกจะต้องสร้างเป็นกำแพงล้อมรอบเอาไว้

4. ทางเดินภายในจะสร้างให้วกวนอย่างไรก็ได้ และทางออกจะมีหรือไม่มีก็ได้ถ้าไม่มีทางออก หลังโปรแกรมเดินหาทางออกจนทั่วแล้วไม่พบ จะแสดงข้อความบอกว่าไม่สามารถหาทางออกจากเขาวงกตได้

ตัวอย่างเขาวงกตที่สร้างขึ้นแสดงในรูปที่ 3

ข้อมูลของทางเดินที่ต้องเก็บระหว่างเดินในเขาวงกต

การเดินหาทางออกจากเขาวงกต นั้น ณ จุด ๆ ใด ก่อนจะตัดสินใจเดินทางต่อไปจุดอื่นจะต้องเก็บข้อมูลของจุด ๆ นั้นเอาไว้ในสแต็กก่อน เพราะข้อมูลเหล่านี้จะบอกสถานะของเส้นทาง ซึ่งจะใช้ตัดสินใจอีกครั้งเมื่อเดินถอยหลังกลับมา และข้อมูลที่จำเป็นต้องเก็บคือตำแหน่งของจุด ๆ นั้น และทิศทางที่เราเลือกเดินในครั้งต่อไป ดังนั้นสแต็กก็

```
/* Direction for move forward */
enum direction {N,E,S,W};
struct step
{
        int rowoffset;
        int coloffset;
};

/* Offset distance for move forward */
struct step movestep [W+1] =
{ {-1, 0}, {0, 1},
  { 1, 0}, {0,-1} };

/* Structure for keep information */
struct positioninfo
{
        int row;
        int col;
        enum direction dir;
};
```

รูปที่ 4 แสดงทิศทาง 4 ทิศ และระยะทางที่ต้องปรับ รวมทั้งข้อมูลในแต่ละจุดที่จะต้องเก็บ

```
/* Setup the information stack */
posinfostack ()
{
        Stset(&stakOposfinfo);
        Stitemsize (sizeof (struct positioninfo));
}
```

รูปที่ 5 การกำหนดสแต็กสำหรับใช้งานในโปรแกรมเขาวงกต

จะใช้เก็บข้อมูลของจุดแต่ละจุดนั่นเอง โดยแพคเกจสแต็กจะถูกกำหนดค่าต่าง ๆ ผ่านในฟังก์ชัน posinfostack ( ) ในรูปที่ 5

การเดินในเขาวงกต

การเดินจากจุด ๆ หนึ่งไปยังอีกจุดหนึ่งนั้นในโปรแกรม สามารถจะเลือกเดินได้ถึง 4 ทิศทาง คือ ทิศเหนือ (N), ตะวันออก (E), ใต้ (S) และตะวันตก (W) ทิศทางทั้งสี่ทิศจะตั้งเป็น enum (ในรูปที่ 4) และแต่ละทิศทางการเปลี่ยนตำแหน่งแถวและคอลัมน์จะเหมือนกัน ซึ่งการเปลี่ยนแปลงตำแหน่งจะใช้อะเรย์ชื่อ movestep ในตัวแปรอะเรย์จะเก็บระยะทางที่จะต้องเปลี่ยนของแถวและคอลัมน์ตามทิศทางที่จะเดิน เช่น ถ้าเดินไปทิศเหนือ movestep [N] จะมีค่าเป็น -1, 0 ดังนั้นตำแหน่งแถวและคอลัมน์จึงสามารถจะหาได้จากการคำนวณตำแหน่งแถวและคอลัมน์เดิมบวกด้วยขนาดหรือระยะในแถวและคอลัมน์ที่จะต้องเปลี่ยนดังรูปที่ 6

พบทางตัน

เมื่อโปรแกรมเดินวนจนพบทางตัน ซึ่งหมายถึงทางแยกทั้งหมดคือกำแพง หรือเป็นทางที่เคยเดินผ่านมาแล้ว เส้นทางที่เคยเดินผ่านมาแล้วนั้นโปรแกรมจะเปลี่ยนจุดเหล่านั้นจากช่องว่างให้เป็นจุด '.' แทน ดังนั้นเมื่อเดินเข้าไปจนพบว่าทางแยกทั้งสามคือ ซ้าย ขวา และหน้าเป็นทางที่เดินไม่ได้ โปรแกรมจะเริ่มถอยหลังกลับ (ทางข้างหลังก็เดินไม่ได้ เนื่องจากเป็นทางที่เดินผ่านมาแล้ว) โดยทำการดึงค่าข้อมูลที่เคยเก็บเอาออกจากสแต็ก แต่จะต้องดึงเอาข้อมูลของจุดปัจจุบัน (ที่กำลังยืนอยู่) ทิ้งออกไปก่อน แล้วจึงดึงเอาข้อมูลของจุดที่จะถอยหลังออกมา แล้วเริ่มพิจารณาจากข้อมูลที่เก็บเอาไว้ว่าเคยเดินไปในทิศทางใดแล้วบ้างและตัดสินใจเดินต่อไป

```
nextrow = row + movestep [N] .row offset;
nextcol = col + movestep [N] .col offset;
```

รูปที่ 6

เขาวงกตไม่มีทางออก

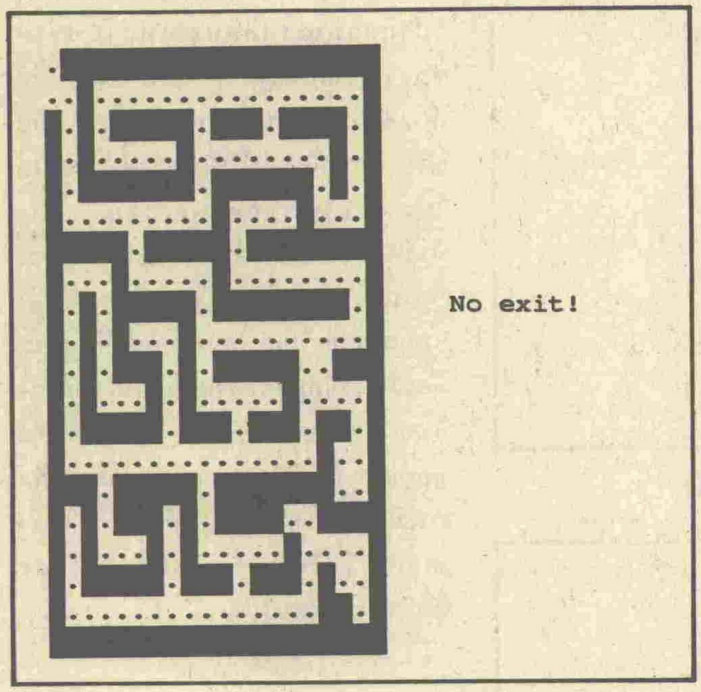
ถ้าทางเดินในเขาวงกตไม่มีทางออก โปรแกรมจะสามารถตรวจสอบได้เอง เนื่องจากโปรแกรมจะเดินถอยหลังกลับมาจุดเดิมตลอดจนกระทั่งถึงจุดเริ่มต้น ซึ่งจะไม่มีข้อมูลในสแต็กให้ดึงออกพิจารณาอีกแล้ว โปรแกรมก็จะหยุดทำงาน

เขาวงกตมีทางออก

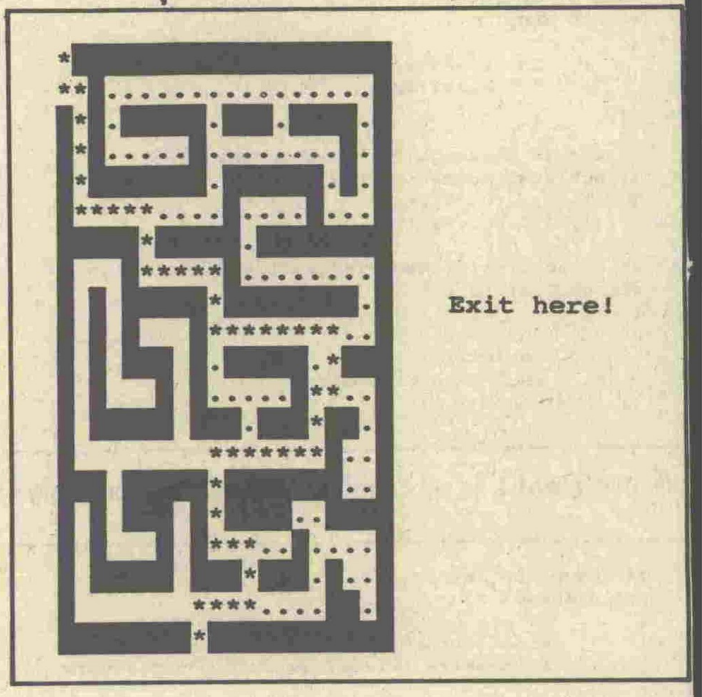
ถ้าทางเดินในเขาวงกตมีทางออก ข้อมูลของจุดต่าง ๆ ที่ถูกเดินผ่านและค้างอยู่ในสแต็กจะแสดงถึงเส้นทางเดินจากจุดเริ่มต้นถึงทางออก ซึ่งในโปรแกรมจะแสดงเส้นทางนี้ให้เห็นโดยจะพิมพ์ตัวอักษร "*" เอาไว้บนทางเดิน ดังแสดงในรูปที่ 8

สำหรับตัวโปรแกรม การหาทางออกจากเขาวงกตแสดงเอาไว้ในรูปที่ 9 ใช้กับเทอร์โบซี

รูปที่ 7 เขาวงกตที่ไม่มีทางออก โปรแกรมจะเดินหาทางออกจนทั่วและจะพิมพ์ผลลัพธ์เมื่อหาทางออกไม่ได้



รูปที่ 8 ทางเดินเขาวงกตที่มีทางออก โปรแกรมจะเดินหาทางออกเมื่อพบจะพิมพ์เส้นทางที่ถูกต้องออกมาให้



รูปที่ 9 โปรแกรม maze.c

```
#include <stdio.h>
#include <string.h>
#include "stackdef.h"


#define MAXCOL    20                /* Max row of maze */
#define MAXROW    20                /* Max column of maze */
#define SCRROW     1                /* Screen row for display */
#define SCRCOL     1                /* Screen column for display */
#define WALL      '▒'     /* Char 219 represent the wall */
#define OLDPATH   '.'               /* Char '.' represent the old path */
#define EXITPATH  '*'               /* Char '*' represent the exit path */

/* Direction for move forward */
enum direction {N,E,S,W};
struct step
{
        int rowoffset;
        int coloffset;
};

/* Offset distance for move forward */
struct step movestep [W+1] =
{ {-1, 0}, {0, 1},
  { 1, 0}, {0,-1} };
```

รูปที่ 9 (ต่อ)

```
/* Structure for keep information */
struct positioninfo
{
        int row;
        int col;
        enum direction dir;
};

/* Stack of information in each position */
struct STACK stakOfposfinfo;

/* Maze uses two dimention array */
char maze [MAXROW][MAXCOL+1];

/* Make the maze */
makemaze ()
{
        FILE *f;
        int i;

        f=fopen ("maze.dat","r");
        for (i=0 ; i<20;i++)
                {
                fscanf (f,"%21c",maze[i]);
                maze[i][MAXROW]='\0';
                }
        fclose (f);
}

/* Display the maze to screen */
dispmaze ()
{
        int i;

        for (i = 0; i < MAXROW; i++)
                {
                gotoxy (SCRCOL,SCRROW+i);
                printf ("%s",maze[i]);
                }
}
/* Make every point that uses to pass as oldpath */
markoldpath (row,col)
int row,col;
{
        maze [row][col] = OLDPATH;
}

/* Find the next step for move */
int nextstep (pos)
struct positioninfo *pos;
{
        int nextrow,nextcol;
```

รูปที่ 9 (ต่อ)

```
        nextrow = pos->row + movestep [pos->dir].rowoffset;
        nextcol = pos->col + movestep [pos->dir].coloffset;
        if (maze [nextrow][nextcol] != WALL &&
            maze [nextrow][nextcol] != OLDPATH &&
            nextrow < MAXROW && nextcol < MAXCOL &&
            nextrow >= 0 && nextcol >= 0)
            return 1;
        else
            return 0;
}

/* Move forward to the point that can move,
  and make present point to old point */
move (pos)
struct positioninfo *pos;
{
        if (pos->dir < W)
                {
                do
                        {

                        /* if can move in this direction */
                        if (nextstep (pos))
                            {
                            pos->row += movestep [pos->dir].rowoffset;
                            pos->col += movestep [pos->dir].coloffset;
                            pos->dir = N;
                            return 1;
                            }
                        else
                                /* if not can't move change direction */
                                pos->dir++;
                        }
                while (pos->dir<=W);
                }
        /* this point is can't move forward */
        return 0;
}

/* Pop the information of previous point */
int backtrack (pos)
struct positioninfo *pos;
{
        if (Stlength ()!=0)
                {
                Stpop (pos);
                return 1;
                }
        else
                return 0;
}
```

รูปที่ 9 (ต่อ)

```
/* Display status of point */
disppos (pos,status)
struct positioninfo *pos;
char status;
{
        gotoxy (pos->col+SCRCOL,pos->row+SCRROW);
        putchar (status);
}

/* Check exit the maze, when move
to MAXROW - 1 */
int exitmaze (pos)
struct positioninfo *pos;
{
        if (pos->row == MAXROW-1)
                return 1;
        else
                return 0;
}

/* Display message */
dispmsg (msg)
char *msg;
{
        gotoxy (30,10);
        printf ("%s",msg);
}

/* Find the exit path */
findpath (pos)
struct positioninfo *pos;
{
        while (!exitmaze (pos))
                {
                Stpush (pos);
                markoldpath (pos->row,pos->col);
                disppos (pos,OLDPATH);
                if (!move (pos))
                   {
                        /* can't move then cancell current point */
                        backtrack (pos);
                   /* and go back to previous point */
                        if (!backtrack (pos))
                                {
                                dispmsg ("No exit!");
                                return 0;
                                }
                   }
                }
        disppos (pos,'*');
        exitpath ();
        dispmsg ("Exit here!");
        return 1;
```

รูปที่ 9 (ต่อ)

```
}

/* Make the start point for move at row 0 ,
column 0 and direction at North */
startpos (pos)
struct positioninfo *pos;
{
        pos->row = 0;
        pos->col = 0;
        pos->dir = N;
}

/* Display the exit path */

exitpath ()
{
struct positioninfo pos;
        do
          {
          Stpop (&pos);
          disppos (&pos,EXITPATH);
          }
        while (Stlength()!=0);
}

/* Setup the information stack */
posinfostack ()
{
        Stset(&stakOfposfinfo);
        Stitemsize (sizeof (struct positioninfo));
}

main ()
{
   struct positioninfo posinfo;

        posinfostack ();
        startpos (&posinfo);
        makemaze ();
        dispmaze ();
        findpath (&posinfo);
}
```

สรุป

มุมภาษาซีได้พัฒนาแพคเกจลิงค์ลิสต์ แพคเกจคิว และแพคเกจสแต็ก ซึ่งเป็นแพคเกจที่สามารถจะนำไปใช้จัดเก็บข้อมูลชนิดใด ๆ ก็ได้ เสร็จตามจุดมุ่งหมายแล้ว โดยแพคเกจเหล่านี้มีคุณสมบัติเป็น Reusable อยู่ในตัวแพคเกจเอง ผู้เขียนได้รวบรวมแพคเกจทั้งสาม รวมทั้งตัวอย่างการนำไปใช้งานทั้งหมดบรรจุอยู่ในซอฟต์เมล์ฉบับนี้แล้ว สำหรับท่านผู้อ่านที่สนใจจะนำไปใช้งาน และผู้เขียนได้ทำตารางสรุปฟังก์ชันต่าง ๆ ของแพคเกจทั้งสามเอาไว้ในรูปที่ 10 ผู้เขียนหวังเป็นอย่างยิ่งว่า แพคเกจทั้งสามคงนำประโยชน์ให้ท่านผู้อ่านได้บ้าง

รูปที่ 10 ตารางสรุปฟังก์ชันของแพคเกจลิงค์ลิสต์ แพคเกจคิว และแพคเกจสแต็ก

ชื่อโมดูล	ชื่อไฟล์ include	ชื่อฟังก์ชัน	การทำงาน
Linklist.c	LLdef.h	LLsetList (new__List)	- กำหนดตัวแปร new__List เพื่อใช้งานกับแพคเกจ linklist
		LLsetsize (size)	- กำหนดขนาดของข้อมูลที่จะใช้บรรจุลงแต่ละ item ของ linklist
		LLmatch (numatching)	- กำหนดฟังก์ชันสำหรับใช้เปรียบเทียบของข้อมูลใน linklist
		LLcheck (lookfor)	- ฟังก์ชันใช้ค้นหาข้อมูลใน linklist ซึ่งใช้ฟังก์ชัน numathing ที่ได้จาก LLmatch มาคำนวณและเปรียบเทียบ
		LLinit (newitem)	- ฟังก์ชันใช้เพิ่มข้อมูลลงใน linklist กรณีเป็นข้อมูลตัวแรกสุด
		LLhead ( )	- เลื่อนตัวชี้ item ไปยังส่วนหัวของ list
		LLtast ( )	- เลื่อนตัวชี้ item ไปยังส่วนท้ายของ list
		LLnext ( )	- เลื่อนตัวชี้ item ไปยัง item ตัวถัดไป
		LLprevious ( )	- เลื่อนตัวชี้ item ไปยัง item ตัวก่อนหน้า
		LLretrieve (olditem)	- ดึงข้อมูลจาก item ปัจจุบันมาใส่ใน olditem
		LLadd (newitem)	- เพิ่มข้อมูล newitem เข้าไปใน link-list ณ ตำแหน่งตัวชี้ item ปัจจุบัน
		LLaddhead (newitem)	- เพิ่มข้อมูล newitem เข้าไปใน link-list ส่วนหัว
		LLdelete ( )	- ลบข้อมูลออกจาก linklist ตรงตำแหน่งตัวชี้ item ปัจจุบัน
		LLlength ( )	- บอกจำนวน item ที่มีอยู่ใน linklist
Queue.c	Quenedef.h	Qset (new Queue)	- กำหนดตัวแปรคิวที่จะให้แพคเกจคิวจัดการ
		Qitemsize (size)	- กำหนดขนาดของข้อมูลที่จะใช้เก็บในคิว
		Qlength ( )	- บอกจำนวนข้อมูลที่อยู่ในคิว
		Qinsert (newitem)	- เพิ่มข้อมูล newitem เข้าไปในคิว
		Qremove (olditem)	- ดึงข้อมูลออกจากคิวมาเก็บไว้ใน olditem
Stack.C	Stackdef.h	Stset (newstack)	- กำหนดตัวแปรสแต็กที่จะให้แพคเกจสแต็กจัดการ
		Stitemsive (size)	- กำหนดขนาดของข้อมูลแต่ละ item ที่จะใช้เก็บในคิว
		Stlength ( )	- บอกจำนวน item ในสแต็ก
		Stpush (newitem)	- เพิ่มข้อมูล newitem เข้าไปในสแต็ก
		Stpop (olditem)	- ดึงข้อมูลออกจากสแต็กเก็บไว้ใน olditem